UNLIMITED

RSRE
MEMORANDUM No. 4323

ROYAL SIGNALS & RADAR ESTABLISHMENT

THE KNUTH-BENDIX COMPLETION ALGORITHM AND ITS SPECIFICATION IN Z

Author: A Smith

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

RSRE MEMORANDUM No. 4323

UNLIMITED

CONDITIONS OF RELEASE

0058626 BR-112412

********** U

********** Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

****

DCAF CODE 090996

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4323

Title: The Knuth-Bendix completion algorithm and its specification in Z

Author: A. Smith

Date: September 1989

Summary

Proving that something is a consequence of a set of axioms can be very difficult. The Knuth-Bendix completion algorithm attempts to automate this process when the axioms are equations. The algorithm is bound up in the area of term rewriting, and so this memorandum contains an introduction to the theory of term rewriting, followed by an overview of the algorithm. Finally a formal specification of the algorithm is given using the language Z [7,8].

CONTENTS

1. Introduction

When carrying out a formal proof, the need frequently arises to carry out substitutions, for example, of a construct for its definition. If what is substituted in a theorem is simpler than the term being substituted, the theorem can be simplified, and in many cases proved. This process of substitution is called **term rewriting**, and the **Knuth-Bendix completion algorithm** is concerned with the **mechanical** application of rewrite rules as an aid to automatic proof.

Any set of equalities may be used to provide the rewrite rules, for example the set E of equations

$e \$ x = x$	(equation 1)
$(inv\ x) \$ x = e$	(equation 2)
$(x \$ y) \$ z = x \$ (y \$ z)$	(equation 3)

define a group. Here, e is a constant (the identity element of the group), *inv* is a function which finds the inverse of any element x, and $ is an infix operator which composes two elements of the group.

These equations may be turned into a set of **rewrite rules**, R

$e \$ x \longrightarrow x$	(rule 1)
$(inv\ x) \$ x \longrightarrow e$	(rule 2)
$(x \$ y) \$ z \longrightarrow x \$ (y \$ z)$	(rule 3)

which may be used from left to right and **never** the other way. This important point is the distinction between an equation and a rewrite rule; an equation can be used in either direction, whereas a rewrite rule can only be used from left to right. Clearly if rewrite rules are to be applied automatically, we would not want both directions of the original equations to be represented as rewrite rules, since this could cause the computer to loop.

For example, from **equation 2**, we have $(inv\ x) \$ x = e$ and $e = (inv\ x) \$ x$ but from **rule 2**, we only have $(inv\ x) \$ x \longrightarrow e$, which reads $(inv\ x) \$ x$ **rewrites** to e.

The variables of the rules, x, y, and z, can be substituted with any well formed expressions made from e, x, y, z, *inv* and $. Such expressions are called **terms**.

For example, from rule 3 $(e \$ z) \$ (x \$ x) \longrightarrow e \$ (z \$ (x \$ x))$, where the term e has been substituted for the variable x of the rule, the term z for y, and the term $x \$ x$ for z.

Rewrite rules are widely used in computing, since rewriting is an ideal task for a computer. Manipulating equations could lead to a computer "looping", since it could use an equation one way, then the other, and so on. Careful choice of rewrite rules can overcome this problem.

The use of rewrite rules instead of equations can cause a loss of power. This may be illustrated by considering the analysis tool **MALPAS**[1, 5] which has an automatic term rewriting engine. This tool can be used to formally verify existing software. In MALPAS, rewrite rules are called replacement rules, and the three group theory rules would be introduced as follows:

```
TYPE group;

FUNCTION inv (group) : group;
INFIX $ (group, group) : group;

CONST e : group;

REPLACE (x : group)  e $ x  BY  x;
REPLACE (x : group)  (inv x) $ x  BY  e;
REPLACE (x, y, z : group)  (x $ y) $ z  BY  x $ (y $ z);
```

MALPAS has an algebraic simplifier which carries out term rewriting in an attempt to simplify expressions. For example, for the expression

$$(x \$ e) \$ y = e \$ (x \$ y) \qquad \text{(expression 1)}$$

the simplifier will yield the expression *true* since both sides can be rewritten to $x \$ y$. This is because the LHS can firstly be rewritten to the term $x \$ (e \$ y)$ using rule 3, and then this term can be rewritten to the term $x \$ y$ using rule 1 on the subterm $e \$ y$. The RHS of expression 1 can be rewritten directly to the term $x \$ y$ using rule 1.

Now suppose the simplifier is called to simplify the expression

$$(inv\ x) \$ (x \$ e) = e \$ e \qquad \text{(expression 2)}$$

Since no rule can be applied to the LHS (such a term is called **irreducible** with respect to the rules), and only rule 1 can be applied to the RHS, the expression $(inv\ x) \$ (x \$ e) = e$ will be produced. However, using the original **equations**, E, namely

$$\begin{array}{ll} e \$ x = x & \text{(equation 1)} \\ (inv\ x) \$ x = e & \text{(equation 2)} \\ (x \$ y) \$ z = x \$ (y \$ z) & \text{(equation 3)} \end{array}$$

we have

$$\begin{array}{ll} & (inv\ x) \$ (x \$ e) \\ = & ((inv\ x) \$ x) \$ e \qquad \text{(using equation 3)} \\ = & e \$ e \qquad \text{(using equation 2)} \end{array}$$

and thus expression 2 should really simplify to *true*.

This loss of power that results from using rewrite rules instead of equations can sometimes be recovered using the **Knuth-Bendix completion algorithm** [3,4]. Suppose a set of equations E is turned into a set of rules R, as in the example, then the algorithm takes R and produces a new set of rules R' such that $t = u$, where t and u are terms, is a consequence of E precisely when both t and u can be rewritten to the same term using R'. This rewriting of t and u is carried out as far as possible, using the rules in any order, and the terms that result are known as the **normal form** of t and u with respect to R'; denoted $t \downarrow R'$ and $u \downarrow R'$. Any term is guaranteed to have a unique normal form with respect to R', which is not necessarily the case with respect to R.

As an example, Knuth-Bendix completion on the three group theory rules, R, gives the ten rules R' as below.

$e \$ x \longrightarrow x$	(1)	$inv\ e \longrightarrow e$	(6)
$(inv\ x) \$ x \longrightarrow e$	(2)	$inv(inv\ x) \longrightarrow x$	(7)
$(x \$ y) \$ z \longrightarrow x \$ (y \$ z)$	(3)	$x \$ (inv\ x) \longrightarrow e$	(8)
$(inv\ x) \$ (x \$ y) \longrightarrow y$	(4)	$x \$ ((inv\ x) \$ y) \longrightarrow y$	(9)
$x \$ e \longrightarrow x$	(5)	$inv(x \$ y) \longrightarrow (inv\ y) \$ (inv\ x)$	(10)

With these rules, troublesome expressions such as expression 2, namely $(inv\ x) \$ (x \$ e) = e \$ e$, simplify to *true* since $(inv\ x) \$ (x \$ e) \downarrow R' = e$ and $e \$ e \downarrow R' = e$.

The algorithm can also be said to automate deduction, by making it automatic in deciding whether a given expression should simplify to *true* with respect to the original equations, E. This can be illustrated by considering another troublesome expression with respect to R, namely

$$e \$ x = x \$ e \qquad \text{(expression 3)}$$

With respect to R, this expression simplifies to $x = x \$ e$. But with respect to E

	$e \$ x$	
=	$(inv(inv\ x) \$ (inv\ x)) \$ x$	(using equation 2)
=	$inv(inv\ x) \$ ((inv\ x) \$ x)$	(" " 3)
=	$inv(inv\ x) \$ e$	(" " 2)
=	$inv(inv\ x) \$ (e \$ e)$	(" " 1)
=	$inv(inv\ x) \$ (((inv\ x) \$ x) \$ e)$	(" " 2)
=	$(inv(inv\ x) \$ ((inv\ x) \$ x)) \$ e$	(" " 3)
=	$((inv(inv\ x) \$ (inv\ x)) \$ x) \$ e$	(" " 3)
=	$(e \$ x) \$ e$	(" " 2)
=	$x \$ e$	(" " 1)

and so expression 3 should really simplify to *true*, and indeed it does with respect to R'. The above proof using the equations is by no means obvious (nine steps !) and really highlights the advantage of generating R', since expression 3 then simplifies to *true* by the **automatic** application of these rules.

2. Termination of rewrite rules

A bad set of rewrite rules can still cause the computer to loop. Consider the rule

$$x \otimes y \longrightarrow y \otimes x \qquad \text{(rule A)}$$

Since any terms can be substituted for the variables of a rule then

$$y \otimes x \longrightarrow x \otimes y$$

giving the infinite rewrite sequence

$$x \otimes y \longrightarrow y \otimes x \longrightarrow x \otimes y \longrightarrow \ldots.$$

So any set of rewrite rules containing rule A could cause the computer to loop.

A set of rules supplied to the Knuth-Bendix algorithm must not cause this problem. A set of rules that do not cause this problem is called **terminating** [2]. To show termination of a set of rules R, there must exist an ordering $>>$, on terms, which is

1. **transitive**
   $t >> u \wedge u >> v \Rightarrow t >> v$ for all terms t, u, and v

2. **irreflexive**
   $\neg (t >> t)$ for all terms t

3. **closed with respect to substitution**
   $t >> u \Rightarrow S(t, \sigma) >> S(u, \sigma)$
   for all terms t and u, and substitutions σ of the variables for terms
   ($S(t, \sigma)$ denotes the term t after the substitution σ has been carried out)

4. **monotonic**
   $t >> u \Rightarrow f(\ldots,t,\ldots) >> f(\ldots,u,\ldots)$
   (Both t and u form the n^{th} argument for f for some n)
   for all terms t, u, all functions f, and all $n \in 1 \ldots deg\, f$

5. **well founded**
   there are no infinite (strictly descending) sequences of terms
   $t >> u >> v >> w >> \ldots.$

and that $l >> r$ for each rule $l \longrightarrow r$ in R.

With a terminating set of rules R, the successive application of rules to any term t, in any order, will eventually produce an irreducible term.

Property 1 is needed to ensure that as rules are applied one after another, the terms get consistently smaller. Without property 2, there could be the infinite sequence $t \longrightarrow t \longrightarrow t \longrightarrow \ldots$ for some term t. Property 3 is needed because a rule is used in its substituted form. For example, from the rule $e \$ x \longrightarrow x$ then $e \$ (inv\ y) \longrightarrow inv\ y$ can be obtained, and so it is not good enough to know that $e \$ x >> x$; it must also be the case that $e \$ (inv\ y) >> inv\ y$. Property 4 is needed because subterms can be rewritten. For example as $inv(e \$ x) \longrightarrow inv\ x$ by using the rule $e \$ x \longrightarrow x$ on the subterm $e \$ x$ of $inv(e \$ x)$, it is not good enough to know that $e \$ x >> x$; it must also be the case that $inv(e \$ x) >> inv\ x$. Property 5 is needed to ensure that as a term is rewritten to smaller and smaller terms, the rewriting must eventually stop.

Such an ordering >>, is input into the Knuth-Bendix algorithm along with the set of rules. This ordering not only ensures termination of the existing rules, but is needed in forming new rules as described in section 3.1.

The original three group theory rules form a terminating set of rules. This can be shown using the Knuth-Bendix Ordering (KBO) as described in their original paper[4]. In general it considers a simpler term to be smaller than a complicated term. Essentially it regards the complexity of a term to be the number of symbols appearing in the term, but it also weights the function symbols (in the literature, constants are regarded as functions which take no arguments, and so these too are given a weight). It is also necessary to label each function in the form f_i. What follows is the result of letting $f_1 = e$ have weight 1, $f_2 = \$$ have weight 0, and $f_3 = inv$ have weight 0.

The weight $w(t)$, of a term t, is then the number of occurrences of variables in t plus the number of occurrences of e in t. For example the weight of the term $(x \$ y) \$ inv(e \$ x)$ is 4. Let $n(v, t)$ be the number of occurrences of the variable v in the term t. For example $n(x, (y \$ x) \$ x) = 2$.

The ordering >> on the terms, is then defined as follows:

$t >> u$ if and only if

either ($w(t) > w(u)$ **and** $n(v, t) \geq n(v, u)$ for all variables v) (O 1)

or ($w(t) = w(u)$ **and** $n(v, t) = n(v, u)$ for all variables v **and** (O 2)

either $t = inv^m\, v$ **and** $u = v$ for some $m \geq 1$ and variable v (O 2.1)

or $t = inv\, p$ **and** $u = e$ (O 2.2)

or $t = inv\, p$ **and** $u = q \$ r$ (O 2.3)

or $t = p \$ q$ **and** $u = r \$ s$ **and** $p \neq r$ **and** $p >> r$ (O 2.4)

or $t = p \$ q$ **and** $u = p \$ r$ **and** $q \neq r$ **and** $q >> r$ (O 2.5)

or $t = inv\, p$ **and** $u = inv\, q$ **and** $p \neq q$ **and** $p >> q$) (O 2.6)

where p, q, r and s are terms

It can be shown that the above ordering satisfies the 5 conditions required, so now just the rules themselves need to be checked that they obey >>.

rule 1 $e \$ x \longrightarrow x$

$w(LHS) > w(RHS)$ and $n(x, LHS) = n(x, RHS)$. So $LHS >> RHS$ by O 1 .

rule 2 $(inv\, x) \$ x \longrightarrow e$

$w(LHS) > w(RHS)$ and $n(x, LHS) > n(x, RHS)$. So $LHS >> RHS$ by O 1 .

rule 3 $(x \$ y) \$ z \longrightarrow x \$ (y \$ z)$

$w(LHS) = w(RHS)$ and $n(v, LHS) = n(v, RHS)$ for $v = x$, y and z, so O 2 holds.
Also O 2.4 holds with $p = x \$ y$ and $r = x$, since $p >> r$ by O 1 .
So $LHS >> RHS$ by O 2 and O 2.4 .

3. An overview of the Knuth-Bendix algorithm

Using the original set of rewrite rules for a group, it was not possible to establish the equality of *(inv x) $ (x $ e)* and *e $ e*. This was because the intermediate term *((inv x) $ x) $ e* could not be derived using the rewrite rules, because it would have involved using rule 3 backwards. There was no problem when the original equations were used since equations can be used in either direction. This directionality problem is illustrated in figure 1.

3.1 New rules

The Knuth-Bendix completion algorithm replaces all these problem "peaks" as in figure 1 with "troughs" as in figure 3 by introducing new rules. Figure 3 shows how with a new rule

$$(inv\ x)\ \$\ (x\ \$\ y) \longrightarrow y \qquad \text{(rule 4)}$$

in addition to the other rules, both *(inv x) $ (x $ e)* and *e $ e* can be rewritten to the same term *e*, and thus be shown equal using rewrite rules.

The problem therefore is to derive the new rules. Figure 1 shows how the peak term was rewritten using two different rules; rules 2 and 3. Rule 3 was used on the whole term while rule 2 was used on just the subterm *(inv x) $ x*. Knuth and Bendix found that this was characteristic of all problem peaks; that two different rules are used to rewrite the term; one on the whole term, the other on a subterm.

They realised that these peak terms could be generated automatically by **superimposing** the LHS of one rule onto a subterm of the LHS of another (provided this subterm is not simply a variable). It is then guaranteed that the whole term can be rewritten with one rule and a subterm with another. The two resulting terms, known as a **critical pair**, are, if possible, made into a new rule, thus removing the need for a problem peak.

As an example, rule 4 will be constructed from rules 2 and 3. Figure 2 shows the generation of a critical pair, cp_1 and cp_2, from rules 2 and 3. Notice how figure 2 is a general version of figure 1, with the peak of figure 2 using the variable z with that of figure 1 using the constant e. The general case is considered because then other problem peaks of a similar form will be covered by this general case. Thus **general unification**[6] is used to superimpose one rule onto another.

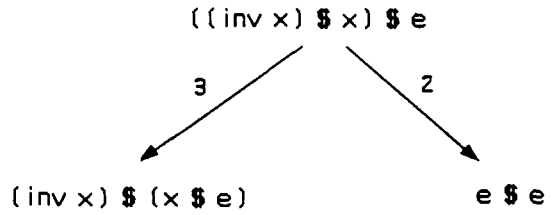

Figure 1 The problem of showing the equality of (inv x) $ (x $ e) and e $ e using the original rewrite rules.

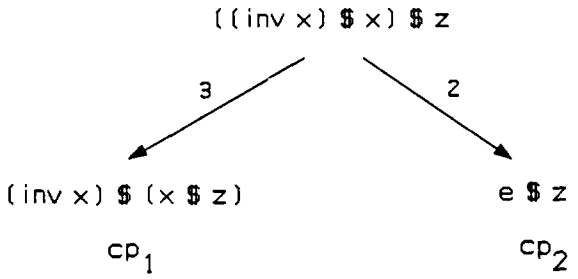

Figure 2 A critical pair of rules 2 and 3

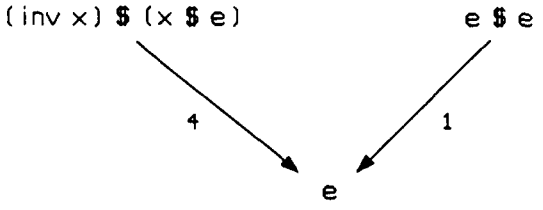

Figure 3 The "peak" of fig 1 replaced by a "trough"

This process of superposition and general unification is best explained by continuing the example which involves rules 2 and 3.

$$(inv\ x)\ \$\ x \longrightarrow e \qquad \text{(rule 2)}$$
$$(x\ \$\ y)\ \$\ z \longrightarrow x\ \$\ (y\ \$\ z) \qquad \text{(rule 3)}$$

The variable x in rule 3 is changed to w

$$(inv\ x)\ \$\ x \longrightarrow e \qquad \text{(rule 2)}$$
$$(w\ \$\ y)\ \$\ z \longrightarrow w\ \$\ (y\ \$\ z) \qquad \text{(rule 3)}$$

so that the rules have no common variable. This in no way changes the meaning of a rule since the variables of a rule are local to that rule and so may be changed.

We want to superimpose the LHS of rule 2 onto the subterm $w\ \$\ y$ of the LHS of rule 3. So general unification is carried out on the terms $(inv\ x)\ \$\ x$ and $w\ \$\ y$. This means finding the most general substitution of the variables for terms that will make both terms the same. In this case it is the substitution

$$\{\ w \mapsto inv\ x, y \mapsto x\ \}$$

where $w \mapsto inv\ x$ means replace w by $inv\ x$. This substitution transforms both terms to

$$(inv\ x)\ \$\ x \qquad \text{(term 1)}$$

Also it is the **most general** substitution in the sense that for any other substitution which makes the two terms the same, say $\{\ x \mapsto e, w \mapsto inv\ e, y \mapsto e\ \}$, then the term that this gives, $(inv\ e)\ \$\ e$, can be obtained from term 1 by means of another substitution, $\{\ x \mapsto e\ \}$. The process of general unification can be carried out automatically using the **general unification algorithm** [6].

Having obtained a critical pair, each is rewritten to its normal form with respect to the current set of rules. In the case of the critical pair of figure 2, this means with respect to rules 1, 2 and 3, obtaining the terms

$$(inv\ x)\ \$\ (x\ \$\ z) \qquad \text{and} \qquad z \qquad \text{(A)}$$

If these terms are different, as in this case, they are made into a **new rule** making sure that it obeys the ordering >> that was input into the algorithm. With the KBO described in the last section, it turns out that $(inv\ x)\ \$\ (x\ \$\ z) >> z$ and so the new rule is

$$(inv\ x)\ \$\ (x\ \$\ z) \longrightarrow z$$

Thus rule 4 has been derived since z can be changed for y. The current set of rules is thus

$e \$ x \longrightarrow x$	(rule 1)
$(inv\ x) \$ x \longrightarrow e$	(rule 2)
$(x \$ y) \$ z \longrightarrow x \$ (y \$ z)$	(rule 3)
$(inv\ x) \$ (x \$ y) \longrightarrow y$	(rule 4)

If, for some other two rules, the terms at stage (A) above are the same, then no new rule is made and the new rule process is started again with another two rules. Also if ever the terms at stage (A) cannot be ordered under >> then the whole algorithm fails. If for all rules and all possible superpositions the terms at stage (A) are the same, then the algorithm terminates successfully with the current set of rules.

3.2 Simplifying the set of rules

Each time a new rule is generated, the entire new set of rules is checked to ensure that each rule is made up only of irreducible terms with respect to the other rules. For example, this is true of the new set of four rules, since for any rule neither side can be reduced with respect to the other rules.

But this is not always the case. If there is a rule where either (or both) sides can be reduced with respect to the other rules, then one of the following can occur:

(1) If the normal forms (with respect to the other rules) of each side are equal then the rule is removed, and the simplification process is started again with the new set of rules. So rules can disappear as well as being created !

(2) If the normal forms (with respect to the other rules) of each side are not equal then the rule is replaced by a new rule made from these normal forms, provided they can be ordered under >>. The simplification process is started again with the new set of rules.

(3) If in (2), the normal forms can not be ordered under >> then the whole algorithm fails.

When all the rules consist only of irreducible terms with respect to the other rules, then the simplification has finished and another new rule is generated from this latest set, as in the last section, and so on.

As an example of the simplification process, the example of the last section will be continued. The current set of four rules do not need simplifying, and this continues to be the case until rule 7 has been generated.

$e \$ x \longrightarrow x$	(rule 1)
$(inv\ x) \$ x \longrightarrow e$	(rule 2)
$(x \$ y) \$ z \longrightarrow x \$ (y \$ z)$	(rule 3)
$(inv\ x) \$ (x \$ y) \longrightarrow y$	(rule 4)
$(inv(inv\ x)) \$ e \longrightarrow x$	(rule 5)
$(inv\ e) \$ x \longrightarrow x$	(rule 6)
$(inv(inv\ x)) \$ y \longrightarrow x \$ y$	(rule 7)

Now rule 5 can be simplified with respect to the other rules, since the LHS of rule 5 can be rewritten using rule 7 to yield the expression $x \$ e$. It turns out that $x \$ e >> x$ and so rule 5 is simplified to $x \$ e \longrightarrow x$, and the set of rules becomes

$e \$ x \longrightarrow x$	(rule 1)
$(inv\ x) \$ x \longrightarrow e$	(rule 2)
$(x \$ y) \$ z \longrightarrow x \$ (y \$ z)$	(rule 3)
$(inv\ x) \$ (x \$ y) \longrightarrow y$	(rule 4)
$x \$ e \longrightarrow x$	(rule 5)
$(inv\ e) \$ x \longrightarrow x$	(rule 6)
$(inv(inv\ x)) \$ y \longrightarrow x \$ y$	(rule 7)

3.3 Some points about the algorithm

(1) For a finite set of rules, the complete set could be infinite and so the algorithm will never terminate (unless it fails because of >>).

(2) The algorithm is very dependent on the ordering >>. For the same set of rules, the algorithm can succeed with one ordering, and not with another.

(3) As described in section 2, the algorithm cannot handle rules of the form $x \otimes y \longrightarrow y \otimes x$, since any rewrite system that contains such a rule will not be terminating. Another example is the rule $x @ (y @ z) \longrightarrow y @ (x @ z)$. Rules of this form are known as permutative rules, because one side can be obtained from the other by a simple permutation of the variables. There are ways of dealing with some of these rules, with different techniques needed for different situations. These techniques are beyond the scope of this report, which concerns itself only with the original Knuth-Bendix algorithm as described in [4].

4. A specification of the Knuth-Bendix completion algorithm in Z

Throughout the specification, examples from the original three group theory rules will be given.

$e \$ x \longrightarrow x$	(rule 1)
$(inv\ x) \$ x \longrightarrow e$	(rule 2)
$(x \$ y) \$ z \longrightarrow x \$ (y \$ z)$	(rule 3)

4.1 Terms

The set of variables *VAR* must include not only the variables contained in the initial set of rewrite rules, but all the extra variables needed during the construction of new rules. For example, the rules above use the variables x, y and z and so these are in *VAR*. Also, when constructing rule 4 the variable w was used (see section 3.1) and so w must be in *VAR*. Other variables are used during the construction of the other rules and so these too must be in *VAR*.

$[VAR]$

The set of functions F together with the variables make up the set of terms. Any constants are regarded as functions which take no arguments. For example, $F = \{ e, inv, \$ \}$.

$[F]$

The degree of each function is the number of arguments it takes. For example, $deg\ e = 0$, $deg\ inv = 1$ and $deg\ \$ = 2$.

$deg : F \longrightarrow \mathbb{N}$

The set of terms are constructed from VAR and F. For example, $z \$ (inv\ x)$ is in *TERM*.

$[TERM]$

$CONSTANT == \{f : F \mid deg\ f = 0\}$

Constructed_term

$f : F$

$s : seq_1 TERM$

$\#s = deg\ f$

$TERM ::= con\langle\langle CONSTANT \rangle\rangle \mid var\langle\langle VAR \rangle\rangle \mid construct\langle\langle Constructed\_term \rangle\rangle$

4.2 Subterms

Any term t can be represented by a tree structure. Also the points on its tree structure can be labelled with sequences of numbers. For any term t, ***subterm*** t is a function which associates each sequence with the subterm from that point. For example, if t is the term $(inv\ y)\ \$\ (y\ \$\ (inv\ x))$ as in figure 4 then ***subterm*** t is the function

$$\{ \langle\rangle \mapsto t, \langle 1 \rangle \mapsto inv\ y, \langle 2 \rangle \mapsto y\ \$\ (inv\ x), \langle 1,1 \rangle \mapsto y, \langle 2,1 \rangle \mapsto y, \langle 2,2 \rangle \mapsto inv\ x, \langle 2,2,1 \rangle \mapsto x \}$$

$$subterm : TERM \rightarrowtail (seq\ \mathbb{N} \nrightarrow TERM)$$

$$\forall\ t : TERM \bullet$$

$$t \notin ran\ construct \Rightarrow subterm\ t = \{\langle\rangle \mapsto t\}$$

$$t \in ran\ construct \Rightarrow$$
$$subterm\ t = \{\langle\rangle \mapsto t\} \cup$$
$$\bigcup \{\ n : dom\ s_1 \bullet \{\ p : dom\ (subterm\ (s_1\ n)) \bullet \langle n \rangle \frown p \mapsto subterm\ (s_1\ n)\ p\ \}\ \}$$

where

$$s_1 == (construct^{-1}\ t).s$$

The function *replace* replaces a subterm of a term with another term. *replace(t, u, p)* is the result of replacing the subterm of t from the point p with the term u. For example if t is the term *(inv y) $ (y $ (inv x))* as in figure 4, p is the the point $\langle 2,2 \rangle$ and u is the term *e $ z* then *replace(t, u, p)* is the term *(inv y) $ (y $ (e $ z))*.

$$replace : (TERM \times TERM \times seq\ \mathbb{N}) \nrightarrow TERM$$

$$\begin{aligned}
&replace = \lambda\, t, u : TERM; p : seq\ \mathbb{N} \mid p \in dom(subterm\ t) \bullet \\
&\quad \mu\, v : TERM \mid \\
&\qquad (t \in ran\ con \land v = u) \\
&\qquad \lor \\
&\qquad (t \in ran\ var \land v = u) \\
&\qquad \lor \\
&\qquad (t \in ran\ construct \land p = \langle\rangle \land v = u) \\
&\qquad \lor \\
&\qquad\quad t \in ran\ construct \\
&\qquad\quad p \neq \langle\rangle \\
&\qquad\quad v \in ran\ construct \\
&\qquad\quad f_2 = f_1 \\
&\qquad\quad s_2 = s_1 \oplus \{\, hd\ p \mapsto replace(s_1(hd\ p), u, tl\ p) \,\} \\
&\qquad where \\
&\qquad\quad f_1 == (construct^{-1}\ t).f \\
&\qquad\quad s_1 == (construct^{-1}\ t).s \\
&\qquad\quad f_2 == (construct^{-1}\ v).f \\
&\qquad\quad s_2 == (construct^{-1}\ v).s \\
&\quad \bullet\ v
\end{aligned}$$

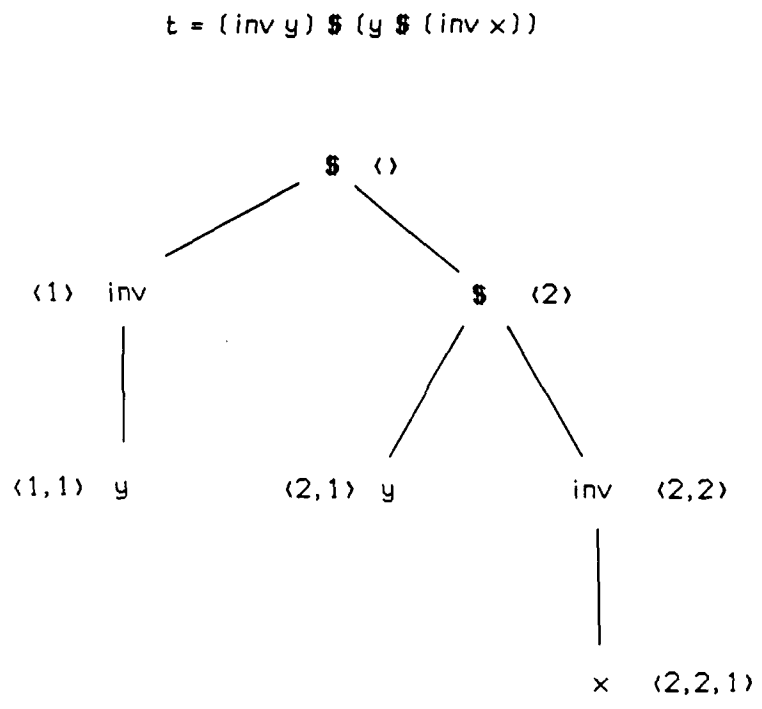

Figure 4 The tree representation of a term

4.3 Substitutions

A substitution is any function from variables to terms, for example $\{ x \mapsto y \$ e,\ z \mapsto e \}$.

$SUBSTITUTION == ran\ var \pfun TERM$

The function S actually performs a substitution on a term. Given a term t and substitution σ, $S(t, \sigma)$ is the result of carrying out σ on t. For any variable v in the domain of σ, all occurrences of v in t are repaced by $\sigma(v)$. Also the substitution of all the variables in the domain of σ that appear in t, are carried out simultaneously. For example, if t is the term $(y \$ y) \$ x$ and σ is the substitution $\{ x \mapsto y \$ e,\ y \mapsto x \}$ then $S(t, \sigma)$ is the term $(x \$ x) \$ (y \$ e)$.

$$
\begin{array}{|l}
\hline
S : (TERM \times SUBSTITUTION) \fun TERM \\
\hline
S = \lambda\, t : TERM; \sigma : SUBSTITUTION \bullet \\
\quad \mu\, u : TERM \mid \\
\qquad (t \in ran\ con \land u = t) \\
\qquad \lor \\
\qquad (t \in ran\ var \land t \in dom\ \sigma \land u = \sigma\, t) \\
\qquad \lor \\
\qquad (t \in ran\ var \land t \notin dom\ \sigma \land u = t) \\
\qquad \lor \\
\qquad\quad t \in ran\ construct \\
\qquad\quad u \in ran\ construct \\
\qquad\quad f_2 = f_1 \\
\qquad\quad \forall\, n : dom\ s_2 \bullet s_2\, n = S(s_1\, n, \sigma) \\
\qquad where \\
\qquad\quad f_1 == (construct^{-1}\, t).f \\
\qquad\quad s_1 == (construct^{-1}\, t).s \\
\qquad\quad f_2 == (construct^{-1}\, u).f \\
\qquad\quad s_2 == (construct^{-1}\, u).s \\
\quad \bullet u \\
\hline
\end{array}
$$

4.4 Terminating rewrite system

As described in section 2, to show termination of a set of rewrite rules there must be an ordering >> on terms which satisfies five conditions. It must then be shown that the rules themselves obey >> . Below are these five conditions, with the definition of a terminating rewrite system afterwards.

$$\_>>\_ : TERM \leftrightarrow TERM$$

$$\forall\, t, u, v : TERM \bullet t >> u \land u >> v \Rightarrow t >> v$$

$$\forall\, t : TERM \bullet \neg\, (t >> t)$$

$$\forall\, t, u : TERM;\ \sigma : SUBSTITUTION \bullet t >> u \Rightarrow S(t, \sigma) >> S(u, \sigma)$$

$$\forall\, t, u : TERM \bullet$$
$$t >> u \Rightarrow (\forall\, v, w : ran\ construct;\ n : \mathbb{N} \mid$$
$$(construct^{-1}\ v).f = (construct^{-1}\ w).f$$
$$n \in dom\ (construct^{-1}\ v).s$$
$$(construct^{-1}\ v).s\ n = t$$
$$(construct^{-1}\ w).s\ n = u$$
$$\bullet\ v >> w)$$

$$\forall\, t : TERM \bullet \exists\, n : \mathbb{N} \bullet t \in dom\ (\ (\_>>\_)^n\ )$$

Given a set of terms and an ordering >> on terms, then a rule is any pair of terms, and a terminating rewrite system is any set of rules that obey >>.

$$RULE == (TERM \times TERM)$$

$$TERMINATING\_REWRITE\_SYSTEM == \mathbb{P}\ \{\ rule : RULE \mid (fst\ rule) >> (snd\ rule)\ \}$$

4.5 Normal form

Given a terminating rewrite system R, then a term t rewrites to another term u, in one step, if and only if there is a rule in R that rewrites a subterm of t (possibly t itself), so that t becomes u. If this is the case then t (--> R) u . For example, if R is the original set of three group theory rules then *(e $ y) $ (e $ x)* rewrites in one step to either *y $ (e $ x)* or *(e $ y) $ x* or *e $ (y $ (e $ x))*.

$$
\begin{aligned}
&\text{--> } == \lambda\, trs : TERMINATING\_REWRITE\_SYSTEM \bullet \\
&\quad \{\, t, u : TERM \mid \exists\, rule : trs;\ p : dom(subterm\ t);\ \sigma : SUBSTITUTION \bullet \\
&\qquad subterm\ t\ p = S(fst\ rule, \sigma) \\
&\qquad u = replace(\ t,\ S(snd\ rule, \sigma),\ p\ ) \\
&\quad \}
\end{aligned}
$$

Given a term t and a terminating set of rules R, then $t \downarrow R$ is the normal form of t with respect to R and is obtained by applying rules from R, to t, one after another until no more apply. In general the normal form is not unique. For example, if R is the original set of three group theory rules then the normal form of *((inv x) $ x) $ e* is either *e* or *(inv x) $ (x $ e)*.

$$
\begin{aligned}
&\downarrow == \lambda\, trs : TERMINATING\_REWRITE\_SYSTEM \bullet \\
&\quad \{\, t, u : TERM \mid \\
&\qquad t \in dom(\text{--> } trs) \Rightarrow u \notin dom(\text{--> } trs) \wedge (t, u) \in ((\text{--> } trs)^{+}) \\
&\qquad t \notin dom(\text{--> } trs) \Rightarrow u = t \\
&\quad \}
\end{aligned}
$$

4.6 Unification

The function ν gives the variables that appear in a term. For example if t is the term $(x \$ y) \$ (x \$ e)$ then $\nu(t)$ is the set $\{x, y\}$.

$$\nu == \lambda\, t : TERM \bullet \{\, x : ran\ var \mid \exists\, p : dom(subterm\ t) \bullet subterm\ t\ p = x \,\}$$

Given two terms t and u, a substitution σ is a most general unifier of t and u if it makes t and u the same and is most general as described in section 3.1. The most general unifier of two terms is not unique; for example the two terms x and y have most general unifier $\{x \mapsto y\}$ or $\{y \mapsto x\}$.

$$mgu : (TERM \times TERM) \leftrightarrow SUBSTITUTION$$

$$\begin{aligned}
&\forall\, t, u : TERM;\ \sigma : SUBSTITUTION \bullet \\
&\quad mgu((t, u), \sigma) \\
&\Leftrightarrow \\
&\quad \nu(t) \cap \nu(u) = \{\} \\
&\quad unify \neq \{\} \\
&\quad \sigma \in unify \\
&\quad \forall\, \tau : unify \bullet \exists\, \rho : SUBSTITUTION \bullet S(\, S(t, \sigma), \rho) = S(t, \tau)
\end{aligned}$$

where

$$unify == \{\, \tau : SUBSTITUTION \mid (\forall\, x : dom\ \tau \bullet x \notin \nu(\tau\, x) \wedge S(t, \tau) = S(u, \tau)) \,\}$$

When *mgu* is used, during the generation of critical pairs, we need to make sure that the two rules have no common variable. For example, in section 3.1, when a critical pair of rules 2 and 3 was being formed, rules 2 and 3 were transformed from *(inv x) $ x —> e* and *(x $ y) $ z —> x $ (y $ z)* to *(inv x) $ x —> e* and *(w $ y) $ z —> w $ (y $ z)*.

$$var\_change : (RULE \times RULE) \leftrightarrow (RULE \times RULE)$$

$$\forall\, rule_1, rule_2, r_1, r_2 : RULE \bullet$$

$$var\_change((rule_1, rule_2), (r_1, r_2)) \Leftrightarrow$$

$$(\nu(fst\ r_1) \cup \nu(snd\ r_1)) \cap (\nu(fst\ r_2) \cup \nu(snd\ r_2)) = \{\}$$

$$\exists\, \sigma, \tau : ran\ var \rightarrowtail ran\ var \bullet$$

$$dom\ \sigma = \nu(fst\ rule_1) \cup \nu(snd\ rule_1)$$

$$dom\ \tau = \nu(fst\ rule_2) \cup \nu(snd\ rule_2)$$

$$S(fst\ rule_1, \sigma) = fst\ r_1$$

$$S(snd\ rule_1, \sigma) = snd\ r_1$$

$$S(fst\ rule_2, \tau) = fst\ r_2$$

$$S(snd\ rule_2, \tau) = snd\ r_2$$

4.7 Critical pairs

As described in section 3.1, critical pairs are used in the generation of new rules. In the schema below, cp_1 and cp_2 are a critical pair. The actual formation of new rules occurs in the function *knuth_bendix* in section 4.9.

Critical_pair

$trs : TERMINATING\_REWRITE\_SYSTEM$

$cp_1, cp_2 : TERM$

$\exists\ rule_1, rule_2 : trs;\ r_1, r_2 : RULE;\ p : seq\ \mathbb{N};\ \sigma : SUBSTITUTION \bullet$

$var\_change((rule_1, rule_2), (r_1, r_2))$

$p \in dom\ (\ subterm\ (fst\ r_1)\ )$

$subterm\ (fst\ r_1)\ p \notin ran\ var$

$mgu(\ (\ subterm\ (fst\ r_1)\ p,\ fst\ r_2\ ), \sigma\ )$

$cp_1 = S(snd\ r_1, \sigma)$

$cp_2 = replace(\ S(fst\ r_1, \sigma),\ S(snd\ r_2, \sigma),\ p\ )$

4.8 Simplifying the set of rules

The function *simplify* corresponds to the simplification of the set of rules as described in section 3.2. Failure has been handled by outputting the empty set of rules. The result of *simplify* is a set of rules where each rule consists only of irreducible terms with respect to the other rules.

$$simplify : TERMINATING\_REWRITE\_SYSTEM \rightarrow TERMINATING\_REWRITE\_SYSTEM$$

$$\forall\ trs_1 : TERMINATING\_REWRITE\_SYSTEM \bullet$$

$$trs_1 = \{\} \wedge simplify\ trs_1 = trs_1$$

$$\vee$$

$$(\ \forall\ rule : trs_1 \bullet$$
$$fst\ rule \notin dom(\ -->(trs_1 \setminus \{rule\})\ )$$
$$snd\ rule \notin dom(\ -->(trs_1 \setminus \{rule\})\ )$$
$$)$$
$$\wedge\ simplify\ trs_1 = trs_1$$

$$\vee$$

$$(\ \exists\ rule : trs_1 \bullet$$
$$v \neq fst\ rule \vee w \neq snd\ rule$$
$$v = w \Rightarrow simplify\ trs_1 = simplify\ trs_2$$
$$v >> w \Rightarrow simplify\ trs_1 = simplify\ (trs_2 \cup \{(v, w)\})$$
$$w >> v \Rightarrow simplify\ trs_1 = simplify\ (trs_2 \cup \{(w, v)\})$$
$$\neg(\ v = w \vee v >> w \vee w >> v\ ) \Rightarrow simplify\ trs_1 = \{\}$$
$$where$$
$$v, w : TERM$$
$$trs_2 : TERMINATING\_REWRITE\_SYSTEM$$
$$---$$
$$trs_2 = trs_1 \setminus \{rule\}$$
$$(fst\ rule, v) \in \downarrow trs_2$$
$$(snd\ rule, w) \in \downarrow trs_2$$
$$)$$

4.9 The algorithm

The function *knuth_bendix* generates a complete set of rules as described in the introduction. It does this recursively by generating new rules from critical pairs, simplifying the set of rules and so on. Once again, failure has been handled by outputting the empty set of rules.

$$
\begin{array}{|l}
knuth\_bendix : TERMINATING\_REWRITE\_SYSTEM \rightarrow \\
\qquad\qquad TERMINATING\_REWRITE\_SYSTEM \\
\hline
\forall\, trs_1 : TERMINATING\_REWRITE\_SYSTEM \bullet \\
\\
\quad trs_1 = \{\} \wedge knuth\_bendix\ trs_1 = trs_1 \\
\\
\quad \vee \\
\\
\quad (\, \forall\, Critical\_pair \mid trs = trs_1 \bullet \\
\qquad t = u \\
\quad \textit{where} \\
\qquad \begin{array}{|l} t, u : TERM \\ \hline (cp_1, t) \in \downarrow trs_1 \\ (cp_2, u) \in \downarrow trs_1 \end{array} \\
\quad) \\
\quad \wedge knuth\_bendix\ trs_1 = trs_1 \\
\\
\quad \vee \\
\\
\quad (\, \exists\, Critical\_pair \mid trs = trs_1 \bullet \\
\qquad t \neq u \\
\qquad t >> u \ \Rightarrow\ knuth\_bendix\ trs_1 = knuth\_bendix\ (\, simplify\ (trs_1 \cup \{(t, u)\})\,) \\
\qquad u >> t \ \Rightarrow\ knuth\_bendix\ trs_1 = knuth\_bendix\ (\, simplify\ (trs_1 \cup \{(u, t)\})\,) \\
\qquad \neg(\, t >> u \ \vee\ u >> t\,) \ \Rightarrow\ knuth\_bendix\ trs_1 = \{\} \\
\quad \textit{where} \\
\qquad \begin{array}{|l} t, u : TERM \\ \hline (cp_1, t) \in \downarrow trs_1 \\ (cp_2, u) \in \downarrow trs_1 \end{array} \\
\quad)
\end{array}
$$

5. Conclusions

Rewrite rules are widely used in computing since rewriting is an ideal task for a computer, and also manipulation of equations could cause a computer to loop, since it could use an equation one way and then the other and so on. Careful choice of rewrite rules can overcome this problem since they can only be used in one direction.

Simply converting equations into rewrite rules by replacing the "equals" sign by a "rewrite" sign can cause a loss of power, with expressions that simplify to *true* using the equations, not simplifying to *true* using the rewrite rules. This report illustrates this point using the MALPAS system which contains an algebraic simplifier which in turn contains a term rewriting engine.

The user could try manually to add extra rules to compensate for this loss of power but runs the risk of adding inconsistencies and non-termination, and even then not being sure that all the power had been recovered. The Knuth-Bendix completion algorithm can, for certain sets of rewrite rules, recover this power automatically without running these risks. If the algorithm terminates successfully then it is guaranteed that the new set of rules it produces is terminating, does not contain inconsistencies, and recovers all the power lost.

The algorithm can also be said to automate deduction, by making it automatic in deciding whether a given expression should simplify to *true* with respect to the original equations. In the MALPAS system, the algorithm would make the algebraic simplifier more powerful.

References

1. B. D. Bramson "Tools for the Specification, Design, Analysis and Verification of software". RSRE report number 87005 (1987)

2. N. Dershowitz "Termination of Rewriting"
pp 69 - 115 in Journal of Symbolic Computation, Vol 3 (1987)

3. A. J. J. Dick "Automated Equational Reasoning and the Knuth-Bendix Algorithm: An Informal Introduction"
Rutherford Appleton Laboratory report number RAL-88-043 (1988)

4. D.E. Knuth and P.B. Bendix "Simple Word Problems in Universal Algebras"
pp 263 - 297 in Computational Problems in Abstract Algebra, ed J. Leech, Pergamon press (1970)

5. MALPAS Intermediate Language (Version 4.0). Rex, Thompson and partners (1987)

6. J. A. Robinson "A Machine-Oriented Logic Based on the Resolution Principle"
pp 23 - 41 in Journal of the Association for Computing Machinery, Vol 12, No 1 (1965)

7. C. T. Sennett "Review of Type Checking and Scope Rules of the Specification Language Z". RSRE report number 87017 (1987)

8. J. M. Spivey "The Z Notation: A Reference Manual"
Prentice-Hall International, Series in Computing Science (1988)

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..... UNCLASSIFIED .....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	Memo 4323		U/C

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
7784000	ROYAL SIGNALS & RADAR ESTABLISHMENT ST ANDREWS ROAD, GREAT MALVERN, WORCESTERSHIRE WR14 3PS
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location

7. Title
THE KNUTH - BENDIX COMPLETION ALGORITHM AND ITS SPECIFICATION IN Z.

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
Smith A			9.1989	23

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement
UNLIMITED

Descriptors (or keywords)

continue on separate piece of paper

Abstract

Proving that something is a consequence of a set of axioms can be very difficult. The Knuth-Bendix completion algorithm attempts to automate this process when the axioms are equations. The algorithm is bound up in the area of term rewriting, and so this memorandum contains an introduction to the theory of term rewriting, followed by an overview of the algorithm. Finally a formal specification of the algorithm is given using the language Z [7,8].